

# TEHNICA DE PROGRAMARE "GREEDY"

## 1. Minimizarea timpului mediu de așteptare

La un ghișeu stau la coadă  $n$  persoane  $p_1, p_2, \dots, p_n$  și pentru fiecare persoană  $p_i$  se cunoaște timpul său de servire  $t_i$ . Să se determine o modalitate de reaşezare a celor  $n$  persoane la coadă, astfel încât timpul mediu de așteptare să fie minim.

De exemplu, să considerăm faptul că la ghișeu stau la coadă  $n = 6$  persoane, având timpii de servire  $t_1 = 7, t_2 = 6, t_3 = 5, t_4 = 10, t_5 = 6$  și  $t_6 = 4$ . Evident, pentru ca o persoană să fie servită, aceasta trebuie să aștepte ca toate persoanele aflate înaintea sa la coadă să fie servite, deci timpii de așteptare ai celor 6 persoane vor fi următorii:

Persoana	Timpul de servire ( $t_i$ )	Timp de așteptare ( $a_i$ )
$p_1$	7	7
$p_2$	6	$7 + 6 = 13$
$p_3$	3	$13 + 3 = 16$
$p_4$	10	$16 + 10 = 26$
$p_5$	6	$26 + 6 = 32$
$p_6$	3	$32 + 3 = 35$
<b>Timpul mediu de așteptare (M):</b>		$\frac{7 + 13 + 16 + 26 + 32 + 35}{6} = \frac{129}{6} = 21.5$

Deoarece timpul de servire al unei persoane influențează timpii de așteptare ai tuturor persoanelor aflate după ea la coadă, se poate intui foarte ușor faptul că minimizarea timpului mediu de așteptare se obține rearanjând persoanele la coadă în ordinea crescătoare a timpilor de servire:

Persoana	Timpul de servire ( $t_i$ )	Timp de așteptare ( $a_i$ )
$p_3$	3	3
$p_6$	3	$3 + 3 = 6$
$p_2$	6	$6 + 6 = 12$
$p_5$	6	$12 + 6 = 18$
$p_1$	7	$18 + 7 = 25$
$p_4$	10	$25 + 10 = 35$
<b>Timpul mediu de așteptare (M):</b>		$\frac{3 + 6 + 12 + 18 + 25 + 35}{6} = \frac{99}{6} = 16.5$

Practic, minimizarea timpului mediu de așteptare este echivalentă cu minimizarea timpului de așteptare al fiecărei persoane, iar minimizarea timpului de așteptare al unei persoane se obține minimizând timpii de servire ai persoanelor aflate înaintea sa!

Pentru a demonstra mai simplu corectitudinea algoritmului, mai întâi vom renumera persoanele  $p_1, p_2, \dots, p_i, \dots, p_j, \dots, p_n$  în ordinea crescătoare a timpilor de servire, astfel încât vom avea  $t_1 \leq t_2 \leq \dots \leq t_i \leq \dots \leq t_j \leq \dots \leq t_n$ . De asemenea, vom presupune faptul că timpii individuali de servire  $t_1, t_2, \dots, t_n$  nu sunt toți egali între ei (în acest caz, problema ar fi trivială), deci există  $i < j$  astfel încât  $t_i < t_j$ . În continuare, presupunem faptul că această modalitate  $P_1$  de aranjare a persoanelor la coadă (o permutare, de fapt) nu este optimă, deci există o altă modalitate optimă  $P_2$  de aranjare  $p_1, p_2, \dots, p_j, \dots, p_i, \dots, p_n$  diferită de cea inițială, în care  $t_j > t_i$  (practic, am interschimbat persoanele  $p_i$  și  $p_j$  din varianta inițială, adică persoana  $p_j$  se află acum pe poziția  $i$  în coadă, iar persoana  $p_i$  se află acum pe poziția  $j$ , unde  $i < j$ ).

În cazul primei modalități de aranjare  $P_1$ , timpul mediu de servire  $M_1$  este egal cu:

$$\begin{aligned} M_1 &= \frac{t_1 + (t_1 + t_2) + \dots + (t_1 + \dots + t_i) + \dots + (t_1 + \dots + t_j) + \dots + (t_1 + \dots + t_n)}{n} = \\ &= \frac{nt_1 + (n-1)t_2 + \dots + (n-i+1)t_i + \dots + (n-j+1)t_j + \dots + 2t_{n-1} + t_n}{n} \end{aligned}$$

În cazul celei de-a doua modalități de aranjare  $P_2$ , timpul mediu de servire  $M_2$  este egal cu:

$$\begin{aligned} M_2 &= \frac{t_1 + (t_1 + t_2) + \dots + (t_1 + \dots + t_j) + \dots + (t_1 + \dots + t_i) + \dots + (t_1 + \dots + t_n)}{n} = \\ &= \frac{nt_1 + (n-1)t_2 + \dots + (n-i+1)t_j + \dots + (n-j+1)t_i + \dots + 2t_{n-1} + t_n}{n} \end{aligned}$$

Comparăm acum  $M_1$  cu  $M_2$ , calculând diferența dintre ele:

$$\begin{aligned} M_1 - M_2 &= \frac{(n-i+1)t_i + (n-j+1)t_j - (n-i+1)t_j - (n-j+1)t_i}{n} = \\ &= \frac{t_i(n-i+1-n+j-1) + t_j(n-j+1-n+i-1)}{n} = \\ &= \frac{t_i(-i+j) + t_j(-j+i)}{n} = \frac{-t_i(i-j) + t_j(i-j)}{n} = \frac{(t_j - t_i)(i-j)}{n} \end{aligned}$$

Deoarece  $i < j$  și  $t_j > t_i$ , obținem faptul că  $M_1 - M_2 = \frac{(t_j - t_i)(i-j)}{n} < 0$  (evident,  $n \geq 1$ ), ceea ce implică  $M_1 < M_2$ . Acest fapt contrazice optimalitatea modalității de aranjare  $P_2$ , deci presupunerea că modalitatea de aranjare  $P_1$  (în ordinea crescătoare a timpilor de servire) nu ar fi optimă este falsă!

Atenție, soluția acestei probleme constă într-o rearanjare a persoanelor  $p_1, p_2, \dots, p_n$ , deci în implementarea acestui algoritm nu este suficient să sortăm crescător timpii de servire, ci trebuie să memorăm perechi de forma  $(p_i, t_i)$ , folosind, de exemplu, un tuplu, iar apoi să le sortăm crescător după componenta  $t_i$ .

```
# functie folosita pentru sortarea crescătoare a persoanelor
# în raport de timpii de servire (cheia)
def cheieTimpServire(t):
    return t[1]

# funcția afișează, într-un format tabelar, timpii de servire
# și timpii de așteptare ai persoanelor
# ts = o listă cu timpii individuali de servire
def afisareTimp(ts):
    print("Persoana\tTimp de servire\tTimp de asteptare")
    # timpul de așteptare al persoanei curente
    tcrt = 0
    # timpul total de așteptare
    tttotal = 0
    for t in ts:
        tcrt = tcrt + t[1]
        tttotal = tttotal + tcrt
        print(str(t[0]).center(len("Persoana")),
              str(t[1]).center(len("Timp de servire")),
              str(tcrt).center(len("Timp de așteptare")), sep="\t")
    print("Timpul mediu de așteptare:", round(tttotal/len(ts), 2))

# timpii de servire ai persoanelor se citesc de la tastatură
aux = [int(x) for x in input("Timpii de servire: ").split()]
# asociem fiecărui timp de servire numărul de ordine al persoanei
tis = [(i+1, aux[i]) for i in range(len(aux))]

print("Varianta inițială:")
afisareTimp(tis)

# sortăm persoanele în ordinea crescătoare a timpilor de servire
tis.sort(key=cheieTimpServire)

print("\nVarianta optimă:")
afisareTimp(tis)
```

Evident, complexitatea algoritmului este dată de complexitatea operației de sortare utilizate, deci complexitatea sa, optimă, este  $\mathcal{O}(n \log_2 n)$ .

Încheiem prezentarea acestei probleme precizând faptul că este o problemă de planificare, forma sa generală fiind următoarea: "Se consideră  $n$  activități cu duratele  $t_1, t_2, \dots, t_n$  care partajează o resursă comună. Știind faptul că activitățile trebuie efectuate

*sub excludere reciprocă (respectiv, la un moment dat, resursa comună poate fi alocată unei singure activități), să se determine o modalitate de planificare a activităților astfel încât timpul mediu de așteptare să fie minim."*

## 2. Planificarea optimă a unor spectacole într-o singură sală

Considerăm  $n$  spectacole  $S_1, S_2, \dots, S_n$  pentru care cunoaștem intervalele lor de desfășurare  $[s_1, f_1), [s_2, f_2), \dots, [s_n, f_n)$ , toate dintr-o singură zi. Având la dispoziție o singură sală, în care putem să planificăm un singur spectacol la un moment dat, să se determine numărul maxim de spectacole care pot fi planificate fără suprapuneri. Un spectacol  $S_j$  poate fi programat după spectacolul  $S_i$  dacă  $s_j \geq f_i$ .

De exemplu, să considerăm  $n = 7$  spectacole având următoarele intervale de desfășurare:

$S_1: [10^{00}, 11^{20})$

$S_2: [09^{30}, 12^{10})$

$S_3: [08^{20}, 09^{50})$

$S_4: [11^{30}, 14^{00})$

$S_5: [12^{10}, 13^{10})$

$S_6: [14^{00}, 16^{00})$

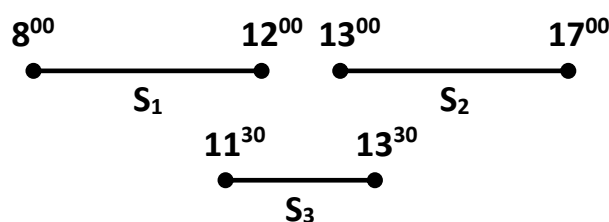
$S_7: [15^{00}, 15^{30})$

Se observă faptul că numărul maxim de spectacole care pot fi planificate este 4, iar o posibilă soluție este  $S_3, S_1, S_5$  și  $S_7$ . Atenție, soluția nu este unică (de exemplu, o altă soluție optimă este  $S_3, S_1, S_5$  și  $S_6$ )!

Deoarece dorim să găsim o rezolvare de tip Greedy a acestei probleme, vom încerca planificarea spectacolelor folosind unul dintre următoarele criterii:

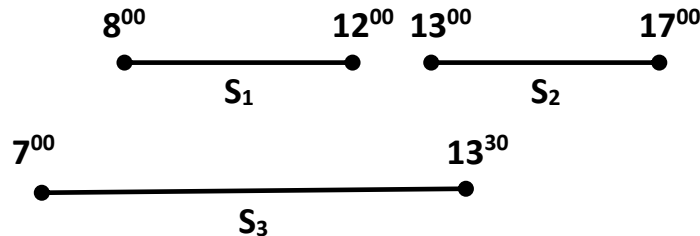
- în ordinea crescătoare a duratelor;
- în ordinea crescătoare a orelor de început;
- în ordinea crescătoare a orelor de terminare.

În cazul utilizării criteriului a), se observă ușor faptul că nu vom obține întotdeauna o soluție optimă. De exemplu, să considerăm următoarele 3 spectacole:



Aplicând criteriul a), vom planifica prima dată spectacolul  $S_3$  (deoarece durează cel mai puțin), iar apoi nu vom mai putea planifica nici spectacolul  $S_1$  și nici spectacolul  $S_2$ , deoarece ambele se suprapun cu spectacolul  $S_3$ , deci vom obține o planificare formată doar din  $S_3$ . Evident, planificarea optimă, cu număr maxim de spectacole, este  $S_1$  și  $S_2$ .

De asemenea, în cazul utilizării criteriului b), se observă ușor faptul că nu vom obține întotdeauna o soluție optimă. De exemplu, să considerăm următoarele 3 spectacole:



Aplicând criteriul b), vom planifica prima dată spectacolul  $S_3$  (deoarece începe primul), iar apoi nu vom mai putea planifica nici spectacolul  $S_1$  și nici spectacolul  $S_2$ , deoarece ambele se suprapun cu el, deci vom obține o planificare formată doar din  $S_3$ . Evident, planificarea optimă este  $S_1$  și  $S_2$ .

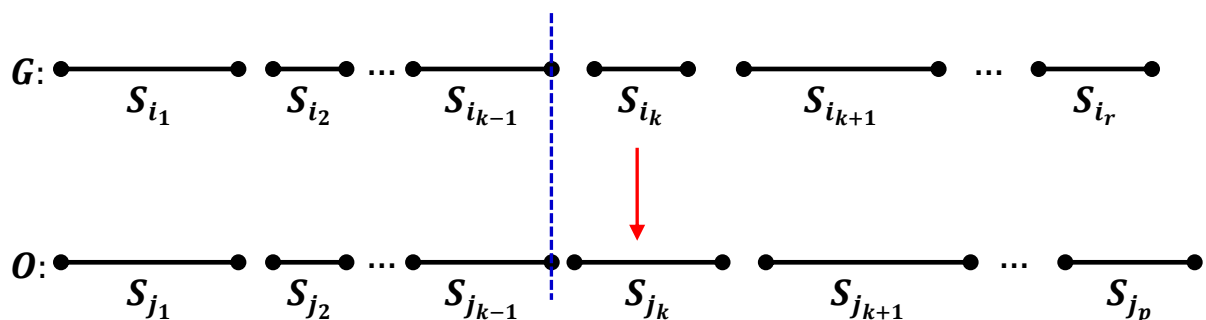
În cazul utilizării criteriului c), se observă faptul că vom obține soluțiile optime în ambele exemple prezentate mai sus:

- în primul exemplu, vom planifica mai întâi spectacolul  $S_1$  (deoarece se termină primul), apoi nu vom putea planifica spectacolul  $S_3$  (deoarece se suprapune cu  $S_1$ ), dar vom putea planifica spectacolul  $S_2$ , deci vom obține planificarea optimă formată din  $S_1$  și  $S_2$ ;
- în al doilea exemplu, vom proceda la fel și vom obține planificarea optimă  $S_1$  și  $S_2$ .

Practic, criteriul c) este o combinație a criteriilor a) și b), deoarece un spectacol care durează puțin și începe devreme se va termina devreme!

Pentru a demonstra optimalitatea criteriului c) de selecție, vom utiliza o demonstrație de tipul *exchange argument*: vom considera o soluție optimă furnizată de un algoritm oarecare (nu contează metoda utilizată!), diferită de soluția furnizată de algoritmul de tip Greedy, și vom demonstra faptul că aceasta poate fi transformată, element cu element, în soluția furnizată de algoritmul de tip Greedy. Astfel, vom demonstra faptul că și soluția furnizată de algoritmul de tip Greedy este tot optimă!

Fie  $G$  soluția furnizată de algoritmul de tip Greedy și o soluție optimă  $O$ , diferită de  $G$ , obținută folosind orice alt algoritm:



Deoarece soluția optimă  $O$  este diferită de soluția Greedy  $G$ , rezultă că există un cel mai mic indice  $k$  pentru care  $S_{i_k} \neq S_{j_k}$ . Practic, este posibil ca ambii algoritmi pot să selecteze, până la pasul  $k - 1$ , aceleași spectacole în aceeași ordine, adică  $S_{i_1} = S_{j_1}, \dots, S_{i_{k-1}} = S_{j_{k-1}}$ . Spectacolul  $S_{j_k}$  din soluția optimă  $O$  poate fi înlocuit cu spectacolul  $S_{i_k}$  din soluția Greedy  $G$  fără a produce o suprapunere, deoarece:

- spectacolul  $S_{i_k}$  începe după spectacolul  $S_{j_{k-1}}$ , deoarece spectacolul  $S_{i_k}$  a fost programat după spectacolul  $S_{i_{k-1}}$  care este identic cu spectacolul  $S_{j_{k-1}}$ , deci  $s_{i_k} \geq f_{i_{k-1}} = f_{j_{k-1}}$ ;
- spectacolul  $S_{j_k}$  se termină după spectacolul  $S_{i_k}$ , adică  $f_{j_k} \geq f_{i_k}$ , deoarece, în caz contrar ( $f_{j_k} < f_{i_k}$ ) algoritmul Greedy ar fi selectat spectacolul  $S_{j_k}$  în locul spectacolului  $S_{i_k}$ ;
- spectacolul  $S_{i_k}$  se termină înaintea spectacolului  $S_{j_{k+1}}$ , adică  $f_{i_k} \leq s_{j_{k+1}}$ , deoarece am demonstrat anterior faptul că  $f_{i_k} \leq f_{j_k}$  și  $f_{j_k} \leq s_{j_{k+1}}$  (deoarece spectacolul  $S_{j_{k+1}}$  a fost programat după spectacolul  $S_{j_k}$ ).

Astfel, am demonstrat faptul că  $f_{j_{k-1}} \leq s_{i_k} < f_{j_k} \leq s_{j_{k+1}}$ , ceea ce ne permite să înlocuim spectacolul  $S_{j_k}$  din soluția optimă  $O$  cu spectacolul  $S_{i_k}$  din soluția Greedy  $G$  fără a produce o suprapunere. Repetând raționamentul anterior, putem transforma primele  $r$  elemente din soluția optimă  $O$  în soluția  $G$  furnizată de algoritmul Greedy.

Pentru a încheia demonstrația, trebuie să mai demonstrăm faptul că ambele soluții conțin același număr de spectacole, respectiv  $r = p$ . Presupunem prin absurd faptul că  $r \neq p$ . Deoarece soluția  $O$  este optimă, rezultă faptul că  $p > r$  (altfel, dacă  $p < r$ , ar însemna că soluția optimă  $O$  conține mai puține spectacole decât soluția Greedy  $G$ , ceea ce i-ar contrazice optimalitatea), deci există cel puțin un spectacol  $S_{j_{r+1}}$  în soluția optimă  $O$  care nu a fost selectat în soluția Greedy  $G$ . Acest lucru este imposibil, deoarece am demonstrat anterior faptul că orice spectacol  $S_{j_k}$  din soluția optimă se termină după spectacolul  $S_{i_k}$  aflat pe aceeași poziție în soluția Greedy (adică  $f_{j_k} \geq f_{i_k}$ ), deci am obține relația  $f_{i_r} \leq f_{j_r} \leq s_{j_{r+1}}$ , ceea ce ar însemna că spectacolul  $S_{j_{r+1}}$  ar fi trebuit să fie selectat și în soluția Greedy  $G$ ! În concluzie, presupunerea că  $r \neq p$  este falsă, deci  $r = p$ .

Astfel, am demonstrat faptul că putem transforma soluția optimă  $O$  în soluția  $G$  furnizată de algoritmul Greedy, deci și soluția furnizată de algoritmul Greedy este optimă!

În concluzie, algoritmul Greedy pentru rezolvarea problemei programării spectacolelor este următorul:

- sortăm spectacolele în ordinea crescătoare a orelor de terminare;
- planificăm primul spectacol (problema are întotdeauna soluție!);
- pentru fiecare spectacol rămas, verificăm dacă începe după ultimul spectacol programat și, în caz afirmativ, îl planificăm și pe el.

Citirea datelor de intrare are complexitatea  $\mathcal{O}(n)$ , sortarea are complexitatea  $\mathcal{O}(n \log_2 n)$ , programarea primului spectacol are complexitatea  $\mathcal{O}(1)$ , testarea spectacolelor rămase are complexitatea  $\mathcal{O}(n - 1)$ , iar afișarea planificării optime are cel mult complexitatea  $\mathcal{O}(n)$ , deci complexitatea algoritmului este  $\mathcal{O}(n \log_2 n)$ .

În continuare, vom prezenta implementarea algoritmului în limbajul Python:

```
# functie folosita pentru sortarea crescătoare a spectacolelor
# în raport de ora de sfârșit (cheia)
def cheieOraSfârșit(sp):
    return sp[2]

# citim datele de intrare din fișierul text "spectacole.txt"
fin = open("spectacole.txt")
# lsp = lista spectacolelor, fiecare spectacol fiind memorat
# sub forma unui tuplu (ID, ora de început, ora de sfârșit)
lsp = []
crt = 1
for linie in fin:
    aux = linie.split("-")
    # aux[0] = ora de început a spectacolului curent
    # aux[1] = ora de sfârșit a spectacolului curent
    lsp.append((crt, aux[0].strip(), aux[1].strip()))
    crt = crt + 1
fin.close()

# sortăm spectacolele în ordinea crescătoare a timpilor de sfârșit
lsp.sort(key=cheieOraSfârșit)

# posp = o listă care conține o programare optimă a spectacolelor,
# inițializată cu primul spectacol
posp = [lsp[0]]
# parcurgem restul spectacolelor
for sp in lsp[1:]:
    # dacă spectacolul curent începe după ultimul spectacol
    # programat, atunci îl programăm și pe el
    if sp[1] >= posp[len(posp)-1][2]:
        posp.append(sp)

# scriem datele de ieșire în fișierul text "programare.txt"
fout = open("programare.txt", "w")
fout.write("Numarul maxim de spectacole: "+str(len(posp))+"\n")
fout.write("\nSpectacolele programate:\n")
for sp in posp:
    fout.write(sp[1]+"-"+sp[2]+" Spectacol "+str(sp[0])+"\n")
fout.close()
```

Pentru exemplul de mai sus, fișierele text de intrare și de ieșire sunt următoarele:

spectacole.txt	programare.txt
10:00-11:20	Numarul maxim de spectacole: 4
09:30-12:10	
08:20-09:50	Spectacolele programate: 08:20-09:50 Spectacol 3
11:30-14:00	

12:10-13:10	10:00-11:20 Spectacol 1
14:00-16:00	12:10-13:10 Spectacol 5
15:00-15:30	15:00-15:30 Spectacol 7

Încheiem prezentarea acestei probleme precizând faptul că este tot o problemă de planificare, forma sa generală fiind următoarea: *"Se consideră  $n$  activități pentru care se cunosc intervalele orare de desfășurare și care partajează o resursă comună. Știind faptul că activitățile trebuie efectuate sub excludere reciprocă (respectiv, la un moment dat resursa comună poate fi alocată unei singure activități), să se determine o modalitate de planificare a unui număr maxim de activități care nu se suprapun."*